Exam, questions

Declarative Programming (University of Melbourne)

The University of Melbourne

Department of Computing and Information Systems

COMP90048

Declarative Programming

SAMPLE EXAM

Exam Duration: Two hours.

Reading Time: Fifteen minutes.

Length: This paper has 3 pages including this cover page.

Authorised Materials: None.

Instructions to Invigilators:

Instructions to Students: Answer all of the questions in the script book provided. Start your answer for each question on a separate page and write the question number at the top of the page. The number of marks for each question (totaling 60) is given; where no breakdown is given for multi-part questions, each part is worth the same number of marks. Minor errors in Haskell and Mercury syntax etc, will be tolerated but you should add comments if you feel it will aid understanding of what you have written.

Calculators: Calculators are not permitted.

Baillieu Library: Exam paper to be held by the library.

Question 1 [5 marks]

What would be printed if the following expressions are typed into `ghci`? Recall `:t` tells `ghci` to just print the type of the expression. If any expressions would result in errors, just give the general nature of the error, for example "type error", rather than detailed error messages.

(a) `:t (<)`

(b) `:t map (+3)`

(c) `map (length < 3) [[1],[1,2,3]]`

(d) `filter (not.(==3)) [1,2,3]`

(e) `let e = head [] in 3`

Question 2 [6 marks]

Assume the following definitions are part of a Mercury module which imports the `list` module.

```
:- pred q2(list(T), T, list(T)).
:- mode q2(in, out, out) is nondet.
:- mode q2(out, in, in) is multi.

q2([H | T], H, T).
q2([H | T], E, [H | NT]) :-
    q2(T, E, NT).
```

What would be computed by the following calls? Give bindings for all the variables in the call (assume all the variables are inferred to be mode `out` for the call) for each solution to the call; if there are no solutions just write "no solutions". If any calls would result in errors, just give the general nature of the error, for example "type error", rather than detailed error messages.

(a) `append(As, Bs, [1,2])`

(b) `q2([1,2], B, Cs)`

(c) `q2(As, [], [1,2])`

Question 3 [7 marks]

Occasionally programmers write code which confuses different physical units. For example, the code may use floats to represent both distance in feet and distance in meters, and the two may be added (which is inappropriate). Explain how an algebraic type system could be used to avoid such errors.

Question 4 [7+5=12 marks]

In Haskell, the less than operator (`<`) can be applied to many different data types, including new types created by the programmer.

(a) Briefly (two or three paragraphs) explain the feature of the Haskell type system which allows this.

(b) We would normally expect `<` to obey certain laws such as transitivity (`x<y` and `y<z` implies `x<z`). Does the Haskell implementation enforce such laws? If so, explain how. If not, explain whether it is important for the programmer to make sure they are obeyed.

Question 5 [5+3=8 marks]

Consider the following Mercury type definition for ternary (three-way) trees:

```
:- type ttree(T)
    --->    nil
    ;       node3(T, ttree(T), ttree(T), ttree(T)).
```

(a) Write a Mercury predicate

```
:- pred map_ttree(pred(A,B), ttree(A), ttree(B)).
```

which is analogous to *map* for lists. Include one or more mode declarations for this predicate so as to make its use as general as possible.

(b) Write a predicate to test if two ttrees have the same "shape" using `map_ttree`.

Question 6 [5+17=22 marks]

Consider the following Haskell equivalent of the Mercury type for ternary trees above:

```
data Ttree t = Nil | Node3 t (Ttree t) (Ttree t) (Ttree t)
```

(a) Write a Haskelll function to test if two ttrees have the same "shape".

(b) Suppose we have a `ttree` of lists and we want to find the shortest list in the tree and the concatenation of all the lists in the tree, in the following order: first the list from the root, then the lists from the left subtree, then those from the middle subtree, then those from the right subtree. Write *either* a Haskell function *or* Mercury predicate which performs this task. Include the appropriate declarations. You must decide the appropriate type(s) to return etc. To obtain maximum marks, your code should use a single traversal over the tree and have $O(N)$ worst case time complexity.